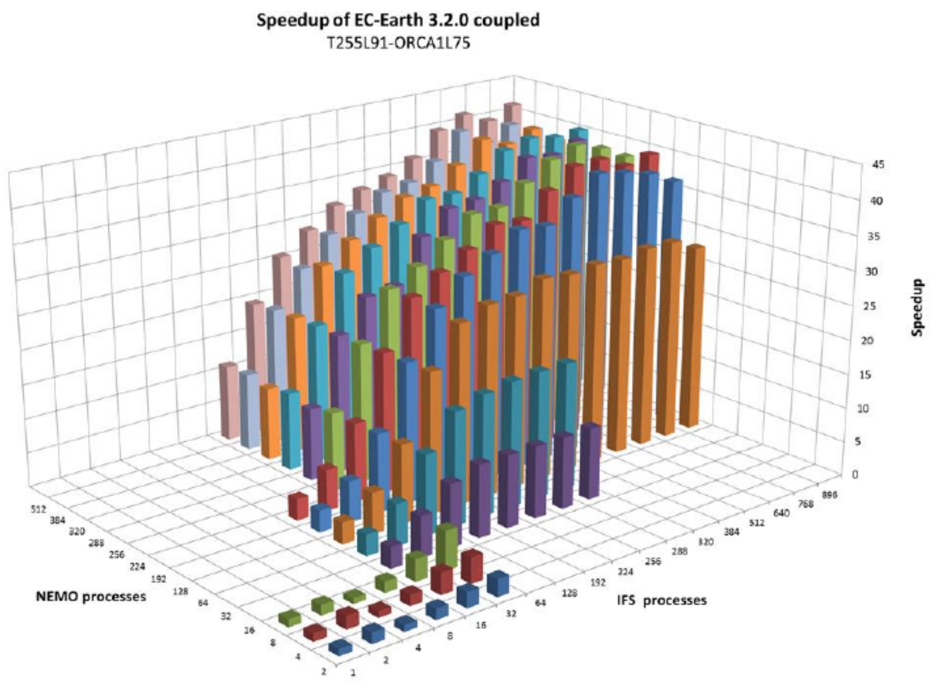
Technical report

HPC application support engineer internship



From: 04/06/2018 to 10/08/2018 - **School**: Polytech Sorbonne - **Speciality**: MAIN - **Academic year**: 2017/2018

Supervised by: Mario ACOSTA

Written by: Fatine BENTIRES ALJ

Résumé

Il existe environ cinq cents super ordinateurs au monde. Un super ordinateur a pour principal but d'effectuer des calculs de hautes performances. Ces calculs de performances sont possibles notamment grâce à **MPI**, une technique de transfert d'information. Ce stage a ainsi porté sur l'utilisation d'un de ces super ordinateurs : **Marenostrum** qui signifie "notre mer".

Au delà de l'utilisation de cet ordinateur, le premier objectif du stage a été l'étude du modèle **Ec-Earth**. Comme l'indique son nom, ce modèle représente la Terre. Il est composé de plusieurs modèles représentants chaque partie de la terre. Ces derniers sont alors couplés entre eux grâce à **OASIS3-MCT**. Les deux modèles étudiés avec précisions lors de ce stage sont le modèle océanique **NEMO** et le modèle atmosphérique **IFS**. Ces deux derniers peuvent être décomposés en 8 étapes. Pour pouvoir terminer ses calculs, NEMO a besoin de données renvoyées par IFS. Or les étapes 4 et 8 d'IFS sont beaucoup plus longues que celles de NEMO. Les temps de calculs s'en voient alors allongés. Le but était ainsi de se familiariser avec les différentes métriques comme le temps de calcul renvoyé par **LUCIA**, un élément de OASIS3-MCT.

Dans un second temps, le but était d'effectuer des tests de **scalabilité informatique**. Ceux-ci reposent sur le lancement de projets avec pour objectif de trouver le meilleur nombre de processeurs à utiliser pour chaque modèle. Après la configuration de la session et l'étude attentive des plots obtenus, j'ai trouvé la meilleure combinaison de processeurs (960 pour IFS et 480 pour NEMO). J'ai par ailleurs créé plusieurs scripts pour automatiser les processus, pour représenter les méthodes **Time** and **Energy to solution** et rédigé de nombreux rapports pour expliquer les différentes caractéristiques du projet.

Pour aller plus loin, il a été décidé d'étudier le problème de l'**équilibrage de charge**. Trouver le meilleur nombre de processeurs est une bonne chose. Cela donne des temps de calculs globaux réduits. Cependant il n'assure pas que les temps obtenus pour les deux modèles sont similaires. En effet, le couplage est effectué après chaque étape. Cela signifie que le modèle le plus rapide doit attendre le plus lent. Ceci doit être pris en considération pour éviter de gaspiller des ressources inutilement. On comprend donc que ce n'est pas suffisant de trouver la meilleure combinaison de processeurs mais qu'il est également nécessaire de trouver le meilleur équilibrage de charge. Cinq méthodes ont alors été créés par la suite. Ainsi, en fonction de la technique employée, le nombre de processeurs optimal diverge. De plus, comme précédemment, les deux options **Time** et **Energy to solution** ont été ajoutées à chaque méthode et de nombreux rapports ont été rédigés.

Enfin, la dernière partie correspond à **l'interface graphique**. Les résultats précédents obtenus, il a été demandé de les présenter de manière plus esthétique. Pour cela, une interface graphique en Python 3 a été développée.

Acknowledges

I would like to express gratitude to the director of the Earth Science department, Kim Serradell, for giving me the chance to join his team. I would like also to thank my supervisor, Mario Acosta, for taking the time to transmit so much knowledge. Obviously I show appreciation to Domingo Manubens and Larissa Batista Leite who helped me to understand autosubmit and to Miguel Castrillo who introduced me a new way of doing the scalability tests. Finally, I greet Blanca Creus from the RH department who helped me to fill all the documentation required.

Contents

Introduction

In the last decade, our understanding of climate change has increased, as society's needs grow for advice and policy. However, whilst there is a general consensus that climate change is an ongoing phenomenon, there remain uncertainties for example, on the levels of greenhouse gas emissions. Increasing the capability and comprehensiveness, in order to represent with ever-increasing realism and detail new scenarios for our future climate, is the only way to reduce these latter uncertainties. However, the increase of this understanding is strongly linked to the amount of computing power and data storage capacity available. Thus, supercomputer are used.

Most applications targeting machines with huge computational power require some degree of rewriting to expose more parallelism, and many of those face severe strong-scaling challenges if they are effectively to achieve this improvement in computational performance, as it is demanded by their scientific goals. There is an ongoing need of support for software maintenance, and tools to manage and optimize work flows all across the infrastructure. As a result, a priority is to take advantage of the parallel resources that we have in supercomputers. However, this is not a trivial task, the typical scalability plots are more difficult to obtain since different components are run in parallel at the same time. This means that a new metric to explore the load balance among components is needed and the question about the best numbers of parallel resources to use, more difficult to answer.

Barcelona Supercomputing Center [1] is the national supercomputing center in Spain. It's specialized in high performance computing and manages **Marenostrum**, one of the most powerful supercomputers in Europe. This supercomputer is used for researching in different areas such as civil Engineering or Medicine. On the most important areas is Earth Science, where scientists run Earth System Models to study the climate change from a few years (prediction) to one hundred or more years (projection). In particular, the model used in the Earth Science department is known as EC-Earth.

EC-Earth is the European global climate community model, based on **IFS**, the world-leading weather forecast model of ECMWF (European Center of Medium Range Weather Forecast) in its seasonal prediction configuration, along with **NEMO**, a state-of-the-art modeling framework for oceanographic forecasting and climate studies, which is developed by the NEMO European Consortium. EC-Earth is one of the models chosen by most of the meteorological centers and research institutes across Europe. Among other purposes, this model is used to provide information to address the regional impacts of climate change, and determine appropriate adaptation and mitigation measures on a more regional basis.

One of the main goals of my work is to achieve a **good scalability** of the EC-Earth model running on different configurations, so that I can evaluate the best efficiency of the execution of EC-Earth for two approaches, time to solution (the best one in terms of execution time) and energy to solution (the best one in terms of efficiency). An other is to create new metrics to explore the **load balance** issue with those two approaches adapted to this part.

Thereby, the report starts with the industrial requirements. Then it expands to the theoretical study with the functionalities of MPI and the components of EC-Earth. Subsequently it explains the technical tools such as the characteristics of a project, the scalability and load balance tests. Finally, it illustrates the results obtained with a graphical interface.

Part I

Industrial requirements

1 Current context

The department which is asking for this project is specialized in Earth-Science. The principle users of this project will be the workers of this department. The project is about the computational time and efficiency. Because of the load balance between IFS and NEMO models, the time of computation is too long. Thus, launching a lot of projects takes too much time. In this purpose, they need to study the interaction between those two models and the optimization of the number of processors used to have the best efficiency possible. A research in this field has already been conducted but it has not been finished.

2 Company requirements

During the first meeting with the Bsc supervisor, a list of principal tasks was introduced. Even if each task is important, the main goal of this internship is to make my own conclusions and compare the different models. The understanding and quality is more important that the quantity.

Figure 1: *List of tasks that must be achieved*

1	Read the documentation to understand how EC-Earth coupling system is working
2	Follow the instructions to configure the session
3	Create a new project and understand which command must be written on the computer
4	Write a documentation about the different folders of a project and their utilities
5	Launch a project following some procedures
6	Modify a Python file in the purpose of having formulas more accurate
7	Write a complete documentation about the launch of a project
8	Launch an important number of tests to find the best combination of processors between NEMO and IFS
9	Create a program that launches the project three times and then takes the average of the three values
10	Modify the Python and Bash files to add some metrics such as the speedup
11	Create the time and energy to solution methods for the scalability part
12	Think about a new way to improve the load balance
13	Create two files for the load balance part
14	Modify the Python and Bash files to add some metrics such as the RTS step 4 and 8 and the efficiency
15	Create the time and energy to solution methods for the unbalance part
16	Create three other files for the load balance part
17	Create a graphical interface
18	Make a review of all the created programs
19	Write a report given all the results
20	Make a review of all the written reports
21	Make a presentation in front of all the department

Part II

Theoretical study

3 Parallel computing: Message Passing Interface library

The ACM Computing Curricula 2005 [3] defined "computing" as follow: "*In a general way, we can define computing to mean any goal-oriented activity requiring, benefiting from, or creating computers. Thus, computing includes designing and building hardware and software systems for a wide range of purposes; processing, structuring, and managing various kinds of information; doing scientific studies using computers; making computer systems behave intelligently; creating and using communications and entertainment media; finding and gathering information relevant to any particular purpose, and so on. The list is virtually endless, and the possibilities are vast.*"

Parallel computing is a type of computation in which a lot of executions of processes are carried out concurrently. The purpose of this type of computation is to divide the work time. Otherwise spoken, parallel computing is the use in a simultaneous way of multiple compute resources in a purpose of solving a computational problem:

- A problem is divided into discrete parts that can be solved concurrently;
- Each part is divided again in a series of instructions;
- Instructions are executed simultaneously on different processors;
- An overall control/coordination mechanism is employed.

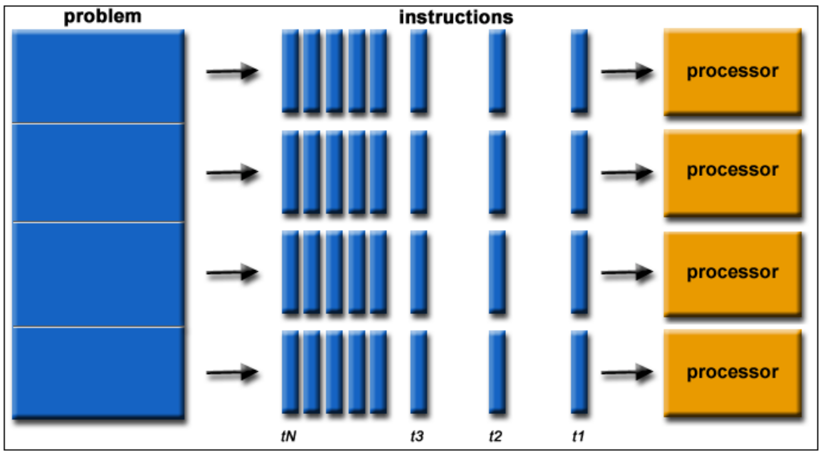


Figure 2: *Parallel computing steps (Picture taken from the "Parallel Computation" documentation written by Blaise Barney, Lawrence Livermore, 2018)*

To do so, some libraries such as Message Passing Interface (MPI) where created in different programming languages.

→ **Remark**: Increasing the number of processors can improve the computational time. However, it takes a lot of memory and resources. Thus, finding the good number of processors is really important. In our case, using a huge number of processors for NEMO and IFS is probably not the best solution. The better way to improve the efficiency is to find the balance between the efficiency of the parallel execution, the execution time and the load balance between the two main components executed in parallel at the same time, IFS and NEMO.

4 EC-Earth

4.1 EC-Earth components

EC-Earth is a project, a consortium and a model system. The consortium is composed of a huge number of academic institutions and meteorological services from different countries in Europe. Thus, the model is quite complete. It is composed of two main components: IFS for the atmospheric model and NEMO for the ocean model. Those two components are coupled using OASIS3-MCT. However, there are some other components:

- *The OASIS3-MCT coupler*: A coupling library used to link the component models by interpolating and exchanging the coupling fields between them.
- *The Integrated Forecasting System* (IFS): This is an operational global meteorological forecasting model developed and maintained by the European Center of Medium-Range Weather Forecasts. The dynamical core of IFS is hydrostatic, two-time-level, semi-implicit, semi-Lagrangian and applies spectral transformations between grid-point space and spectral space. Vertically, the model is discretized using a finite element scheme. A reduced Gaussian grid is used in the horizontal.
- *The Nucleus for European Modeling of the Ocean* (NEMO): It's a framework of oceanographic research, operational oceanography seasonal forecast and climate studies. It's doing a discretization of the 3D Navier-Stokes equations being a finite difference, hydrostatic, primitive equation model, with a free sea surface and a non-linear equation of state. The ocean general circulation model is OPA. OPA is a primitive equation model which is numerically solved in a global ocean curvilinear grid known as ORCA.
- *The Louvain-la-Neuve sea-Ice Model 2/3* (LIM2/3): LIM is a thermodynamic-dynamic sea-ice model directly included in OPA.
- *The hydrological extension of the Tiled ECMWF Surface Scheme for Exchange processes over Land* (HTESSEL): The land and vegetation module which is part of the atmosphere model. It solves the surface energy and water balance taking into account 6 different lands overlying a 4 layer soil scheme. The 6 tiles are: tall vegetation, low vegetation, interception reservoir, bare soil, snow on low vegetation and snow under high vegetation.
- *The Tracer Model 5* (TM5): The chemistry transport model. Describes the atmospheric chemistry and transport of reactive or inert tracers. This component is not used is this work because is not included in the configuration used for the users.
- *The runoff-mapper component*: His purpose is to distribute the runoff from land to the ocean through rivers. It runs using its own binary and is coupled through OASIS-MCT.

Thus, the entire system can be represented thanks to the figures below:

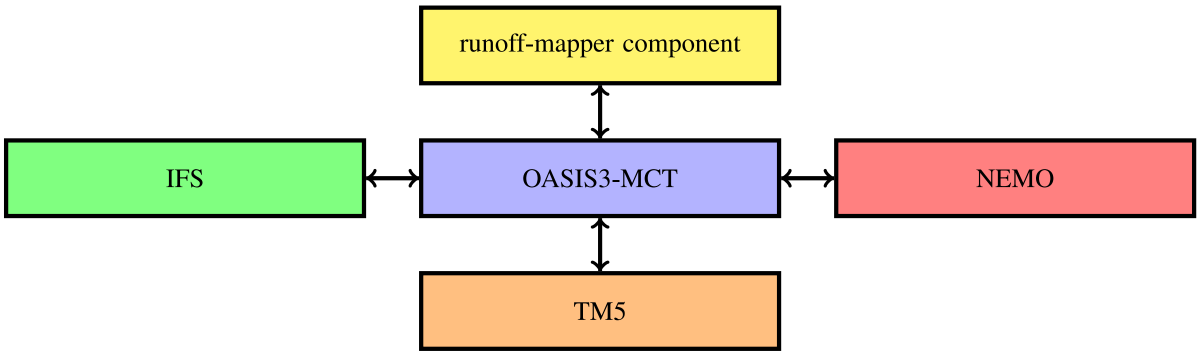


Figure 3: *EC Earth global model*

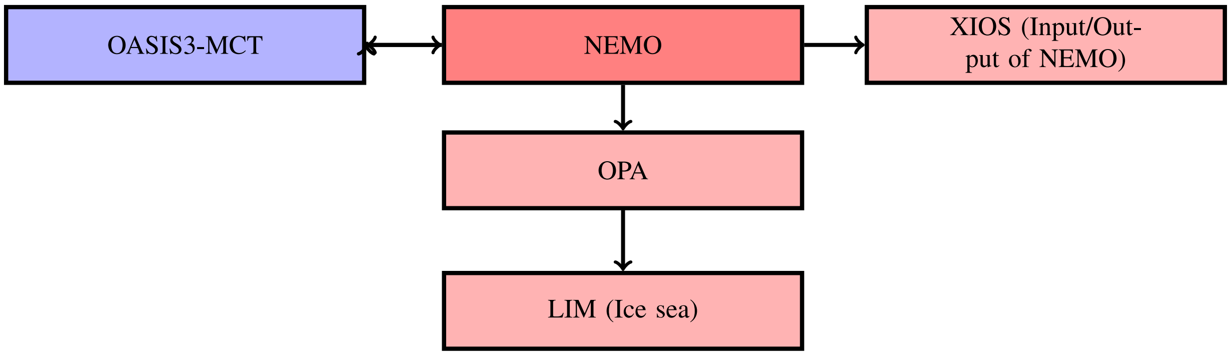


Figure 4: *Ocean model (In red)*

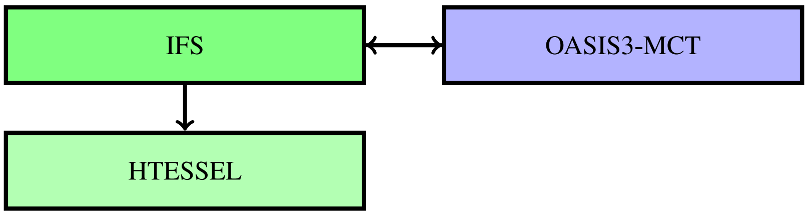


Figure 5: *Atmosphere model (In green)*

$\rightarrow$ **Remark**: A double arrow represents an exchange of information while a single arrow means that the element bellow is included in the element on the top. For example, in the first figure (Figure 3) all components are exchanging informations with OASIS3-MCT. In the second one (Figure 4) , LIM (below) is included in the OPA system (top).

4.2 OASIS3-MCT based coupled system

As explained in the previous section, EC-Earth is coupling different models. A numerical model is an ensemble of discretized equations which mathematically represents one from the several components of the climate system. Thereby, a coupled system is the assembly of some of these numerical models. Thus, we can call "coupling" the representation of the exchanges occurring between components. Exchanges between coupled systems are periodic, with coupling time step periodicity. To be able to perform its own calculations, a model is using information coming at boundaries from another model: at the end of a coupling time step, the target model is waiting for the informations its needs to resume calculations.

OASIS3-MCT library allows to couple climate numerical models. For this purpose, it uses different libraries such as MPI. The MPI communication library ensures the exchange of numerical arrays that hold coupling fields.

However, OASIS interfaced models can be coupled following two different techniques: **sequentially** or **concurrently**. Generally speaking, when a model reaches the beginning of a coupling time step, it needs results of the other coupled model to resume its own calculations. If the results needed by both models are the results of the previous coupling time step, both model can run at the same time (concurrently). If one of the two models needs results of the current coupling time step, this model can be seen as a subroutine of the other one and then models have to run sequentially (each model is waiting for the other one to be able to resume its calculations).

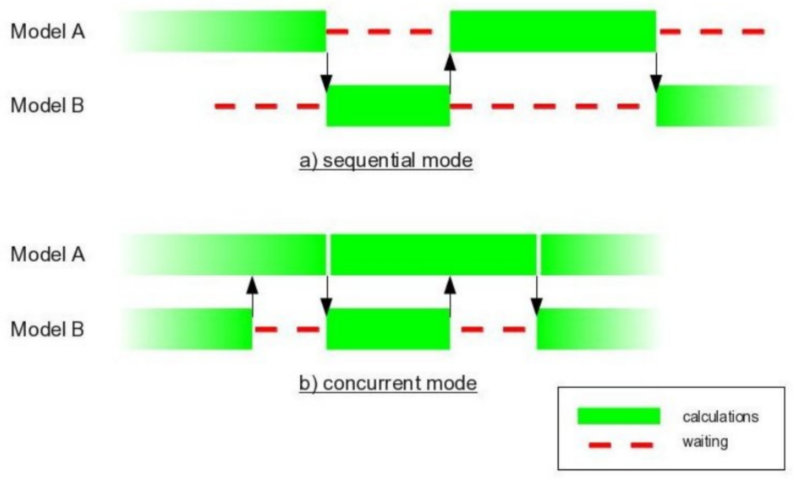


Figure 6: *Two modes of coupling (Picture taken from the "Lucia, load balance tool oasis coupled systems" documentation written by Eric Maisonnave, Arnaud Caubel, 2017)*

For a given model, a coupling time step can be decomposed as follows:

- The model performs its own calculations;
- The coupling library, directly linked to the source model, performs interpolations before sending the coupling fields. The interpolation consists in the **transformation of the value of one variable from the grid of the source component to the grid of the target component**;
- The coupling library sends the coupled variables to other components via MPI communications;
- The coupling library, directly linked to the target model, receives the coupled variables from the source models via MPI communications.

Here we can understand quite easily that the unbalanced computation among the components can affect the final execution time of the coupled model.

4.3 LUCIA performance analysis tool

LUCIA tool [4] is really important. Included in the OASIS3-MCT coupled system, it has the purpose to measure time spent in each phase. Clock time is measured during simulation, saved in log files and finally post-processed to provide clear and concise information. Clock time measures (via MPI_Wtime function) are done in OASIS routines before and after each coupling filed exchange and before and after each interpolation, for each MPI process (if involved in coupling) for each model. When simulation stops, "Lucia" script is launched from the directory where log files were produced. This script calls a Fortran program which reads log files, process and displays on standard output the following quantities:

- *En*: Called "waiting time". This is the time spent by the model (n) sending and receiving MPI messages. More precisely, En measures the **time spent between the beginning and the end of a message sending or receiving**. Since OASIS uses non blocking send (MPI_WaitAll + MPI_ISend), the sending time is the time necessary to write messages into MPI buffer. The receiving time encompasses the time spent to read messages in MPI buffer and the possible load unbalance time between models.
- *Cn*: The time spent by the model (n) to **perform its own calculation and OASIS interpolations**. This time is the complement to En time: Cn + En sum must be equal to the total simulation time used for the analysis.
- *Jn*: Included in Cn. Jitter (Jn) is the **adjustment time needed to wait the moment where all MPI processes are able to send or receive a coupling variable**.

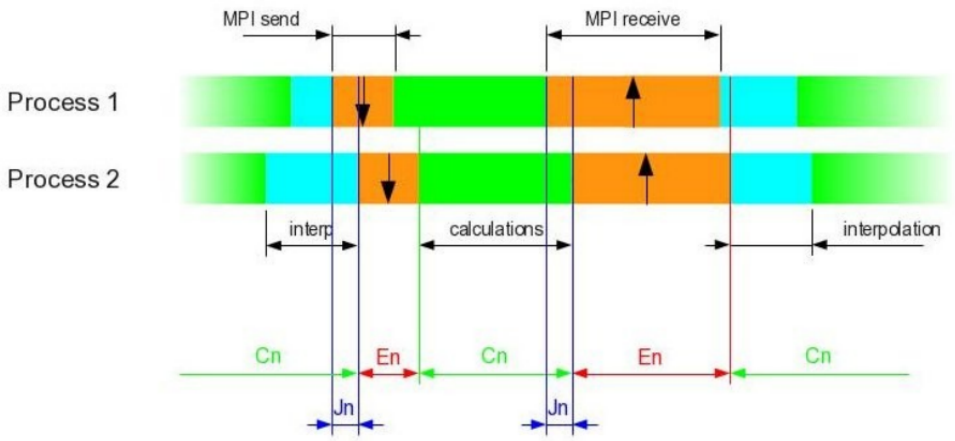


Figure 7: *Lucia's functions (Picture taken from the "Lucia, load balance tool oasis coupled systems" documentation written by Eric Maisonnave, Arnaud Caubel, 2017)*

4.4 Interaction between the models

Like we saw previously, a lot of models are interacting together thanks to the OASIS3-MCT coupling system [6] which includes LUCIA tool. Thereby, load balance between components is achieved **when both components have similar computation and interpolation time** so the communication time is done at the same time and neither IFS nor NEMO have to wait. Unfortunately, IFS takes more time than NEMO to do all the calculations. In fact, like we can see in the figure below, the fourth IFS time step (calculation of the radiation) takes longer. Actually the eight time step, which is not represented here, takes longer also. Thus, the main problem is the difference of time of execution between IFS and NEMO. LUCIA tool will be useful to calculate the different times of execution.

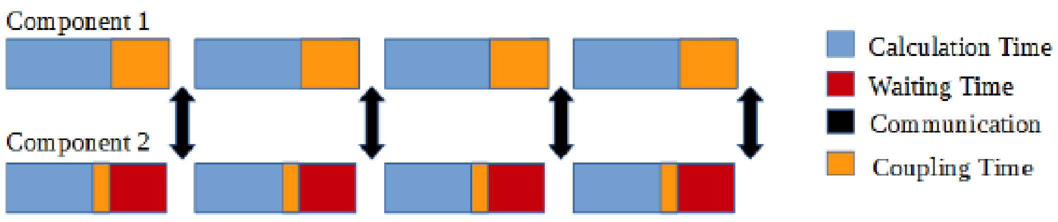


Figure 8: *Four time steps of IFS and NEMO components using the configuration by default in sequential mode (Picture taken from "Performance analysis of EC-Earth 3.2: Load balance" written by Mario Acosta, 2017)*

4.5 EC-Earth metrics

The scalability is the ability of a computer to accept a growing amount of work and to adapt by becoming even more performing. However, to do a scalability test, a project must be created. A project uses a lot of files and thereby a lot of metrics. In this part we are going to present the more important ones.

- ***Tts***: Total time-step, number of steps.

- ***Cn_IFS***: Calculation time of IFS per eight time steps (total time).

$$\frac{(6Rts_{IFS}+1.5Rts_{IFS}+1.86Rts_{IFS})*Tts}{8}$$

- ***Cn_NEMO***: Calculation time of NEMO per eight time steps (total time).

$$Rts_{NEMO} * Tts$$

- ***SYPD***: Simulated Years Per Day.

$$\left(\frac{3month}{executiontimes}\right)\left(\frac{1year}{12month}\right)\left(\frac{3600s}{1h}\right)\left(\frac{24h}{1day}\right) = \frac{21600}{Executiontime(s)}Years/Day$$

- ***RTS_IFS***: The calculation time of IFS on one time step. The 9.3, found empirically is the estimation of the difference of time between a classical time step and a time step that includes the radiation and output time.

$$\frac{Cn_{IFS}*8}{Tts*9.3}$$

- ***RTS_NEMO***: The calculation time of NEMO on one time step.

$$\frac{Cn_{NEMO}}{Tts}$$

- ***IdealCNIFS***: The best Cn_{IFS} that could be obtained if RTS_{IFS} and RTS_{NEMO} were balanced.

$$\frac{Min(RTS_{IFS},RTS_{NEMO})*9.3*Tts}{8}$$

- ***IdealCNIFS***: The best Cn_{NEMO} that could be obtained if RTS_{IFS} and RTS_{NEMO} were balanced.

$$Min(RTS_{IFS},RTS_{NEMO}) * Tts$$

- ***WAITINGIFS***: Total waiting time of IFS.

- ***WAITINGNEMO***: Total waiting time of NEMO.

- ***Unbalance percentage***: Mesure of the unbalance between RTS_{IFS} and RTS_{NEMO}.

$$\frac{RTS_{IFS}*100}{RTS_{NEMO}} - 100$$

- ***Unbalance percentage step four (unbalance4)***: Mesure of the unbalance between the fourth step of IFS and NEMO. This last step is composed of the difference between the regular time step (RTS) of each model to which we add the variation. The variation is the difference of time between the sum of the fourth first steps of IFS and NEMO.

$$RTS_{IFS} + variation$$

- ***Unbalance percentage step eight (unbalance8)***: Mesure of the unbalance between the eighth step of IFS and NEMO. This last step is composed of the difference between the regular time step (RTS) of each model to which we add the variation and the output. The variation is the difference of time between the sum of the eighth steps of IFS and NEMO. The output is the time took to send the results.

$$RTS_{IFS} + variation + output$$

- ***MinTotal***: Fewer time obtainable.

$$1.3 * Min(RTS_{IFS}, RTS_{NEMO}) * \frac{Tts}{8}$$

- ***The speed up***: It's the measure of performance. The execution time base combination is the execution time of the combination that uses fewer MPI processes. The execution time is the execution time of the rest of combination that uses more processes than the base combination.

$$\frac{executiontimebasecombination}{Executiontime}$$

- ***The efficiency***: Is the measure of the uses considering the availabilities. MPI processes base combination is the number of MPI processes of the combination that uses fewer MPI processes. It is used to normalize the data when the base care uses 2 or more processes which is the case of EC-Earth. MPI processes is the number of MPI processes used by a combination to get its speed up.

$$\frac{\frac{MPIprocesses}{MPIprocessesbasecombination}}{Speedup}$$

$\rightarrow$ **Remark**: Some metrics are calculated directly and some others, such as the speed up, are calculated from those first metrics.

Part III

Technical tools

5 EC-Earth configuration

5.1 Creation of a new experiment

First, to start a project we need to ensure that we have a valid SSH key, copy the keys to the HPC machine and RES an account. Once done, we can connect to Marenostrum thanks to the ssh device and then create a new experiment.

Thereby, the project created is the next one :

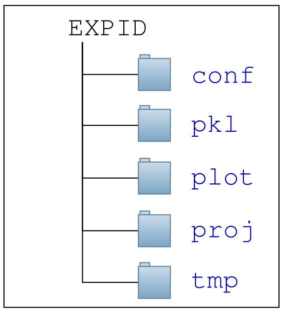


Figure 9: *Configuration of a project*

- *conf*: Controls the creation of the project. In this folder we can find all the setting that needs to be changed depending of the nature of the experimentation such as the computer host or the user id.
- *pkl*: Contains the list of Jobs that must be done to achieve the project.
- *plot* : In this folder we can find all the plots obtained.
- *proj*: It is composed of a lot of folders and files which purposes are to set the Marenostrum and EC-Earth model configuration.
- *tmp*: Contains the messages returned during the execution of the different processes. Those programs have also the purpose to redirect the output files of the components to the good directories.

5.2 Launch of an experiment

At the beginning, 11 steps were required. To automatize the process, I created some files leading to 5 steps:

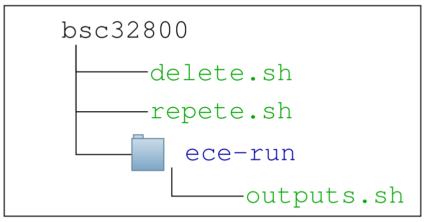


Figure 10: *Files created on Marenostrum*

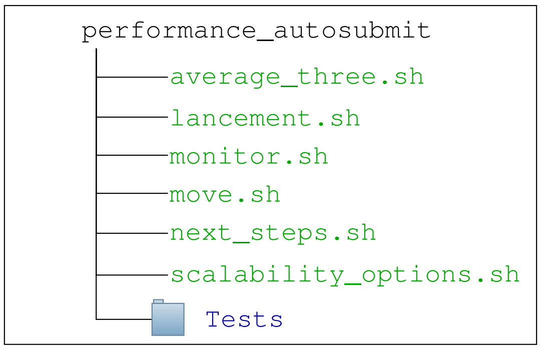


Figure 11: *Files created locally*

In red we can find the commands needed to be executed on Marenostrum and in blue locally:

1. Launch the file lancement.sh. This will launch the performance (file ec-earth_autosubmit_perf.sh) and will monitor the plots. It calls the file "*monitor.sh*" which creates the updated plots.

2. Launch the file repete.sh. This file is made to verify the end of the processes. Each minute the command "squeue" will be executed. **PD** means pending. **R** means running. The processes are ready when there is no appearing line anymore.

3. Launch the file next_steps.sh. This will launch the results (one or three times), will display the plots and will move the files created to a new folder of your choice. This file calls the files "*average_three.sh*" and "*move.sh*". The file "*average_three.sh*" has the purpose to launch the project three times and to take the average of the values. With this method the results are more accurate.

4. Launch the file outputs.sh. This will create the output_lucia files.

5. Launch the file delete.sh. This will execute the last step.

$\rightarrow$ **Remark:** The file ec-earth_autosubmit_perf.sh is composed of two tables. These tables represents the number of processors that will be used for IFS and NEMO to do the calculation and represent the plots.

6 Scalability test

6.1 Classical implementation

The first part of the project was about the scalability test. As it is explained previously, the main goal was to find the best combination in term of processors for NEMO and IFS. To satisfy this aim, I had to create and write several scripts and reports.

First I wrote a report, "*Agnes files documentation*", to explain how the files are interacting together. Then I had to start a project from scratch. Thus, I had to understand which command I needed to use, which files were required and configure the session. This was the longest step since there were a lot of files and no documentation to explain how a project is working. As a result, I wrote a new report, "*complete documentation for the creation of the project*".

Once I understood perfectly how a project is working, I started by modifying the Python file "*ec-earth_autosubmit_perf_display.py*" to have more accurate formulas. In fact, an example is that they were assuming that IFS is taking 9.3 seconds more that NEMO to finish its calculations. However, it's not always the case. I had to use the real values obtained to have more accurate formulas. Then I started to create some bash scripts to automatize the processes like presented in the previous section (5.3). Thus I created "*lancement.sh*". Then I created "*monitor.sh*" to obtain the plots explaining if an experiment is waiting, ready or else.

I created also "*next_steps.sh*" that launches the calculations, displays the results and calls an other file created "*move.sh*" that moves the results to a new folder.

Always in the purpose of being the most efficient possible, I developed three other files, "*delete.sh*", "*outputs.sh*" and "*repete.sh*" but this time on Marenostrum to delete the experiments present in all the folders on the computer, to obtain the Lucia outputs and to repeat the command **squeue** which indicates if a project is pending, running or finished.

Subsequently, after the launch of several projects, I wrote a third report "*Study case: 480 processors for NEMO and 960 for IFS*" which is giving the results for the best combination found.

To go further, I created a last bash script for this part, *average_three.sh* that runs a project three times and then take the average to obtain better values. The main goal about repeating the experiments is to remove the possible variability of some of the executions. We decided with my supervisor to not launch a project more than three times because it's taking too much time.

At the end, I modified the Python file and the Bash file "*ec-earth_autosubmit_perf_display.py/sh*" to obtain a new metric: the speedup. This led to the redaction of three final reports for this part ("*Study case: 128 processors for NEMO and IFS*", "*Results obtained with the modified files*", "*New metric: speed up*") and to the update of the third report ("*Study case: 480 processors for NEMO and 960 for IFS*").

$\rightarrow$ **Remark:** In the annex you can find the work flow of this part with some explanations.

6.2 Case study: Best number of processors for NEMO and IFS

Let's now try to understand the results obtained for a number of processors equal to 480 for NEMO and 960 for IFS. First let's take a look to the CN and RTS files which are the calculation times per eight and one step:

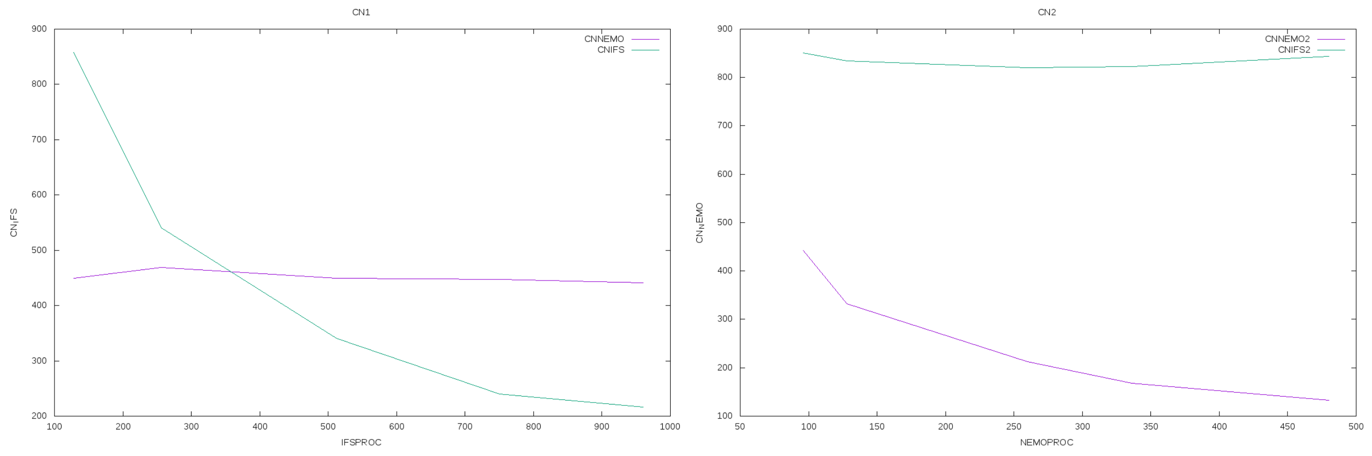


Figure 12: *CN IFS and CN NEMO*

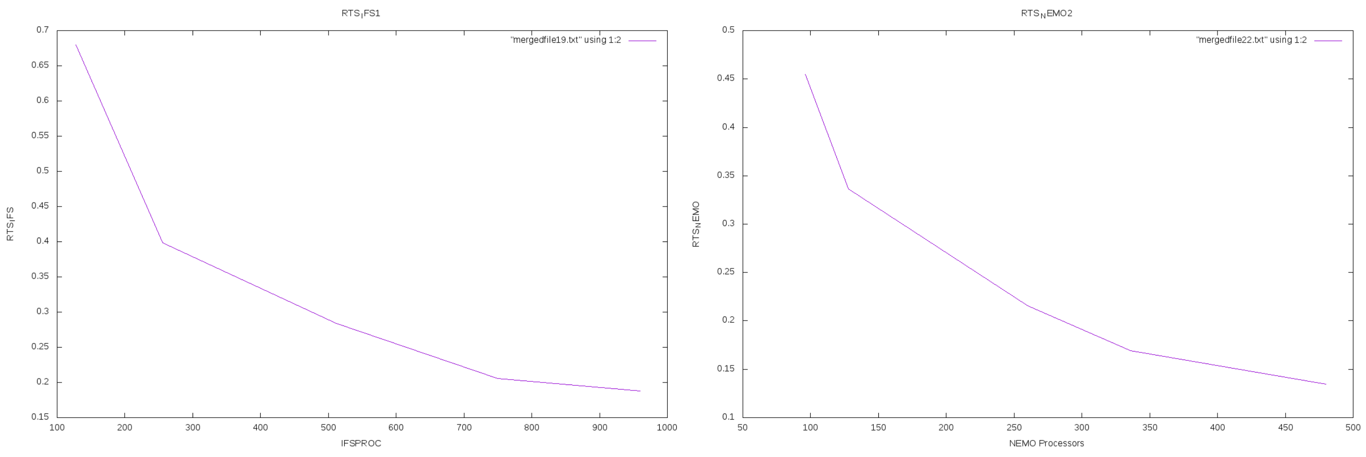


Figure 13: *RTS IFS and RTS NEMO*

We can see that the number of processors for IFS seems optimal since the value of CN and RTS are really low. We can also note that from roughly 350 processors the calculation time for eight steps for IFS is better than for NEMO. However, it seems like the best number of processors for NEMO is beyond 450 processors. We can ask ourself what could have been the best number of CN if NEMO and IFS were balanced. To answer this question we have to understand the following plots.

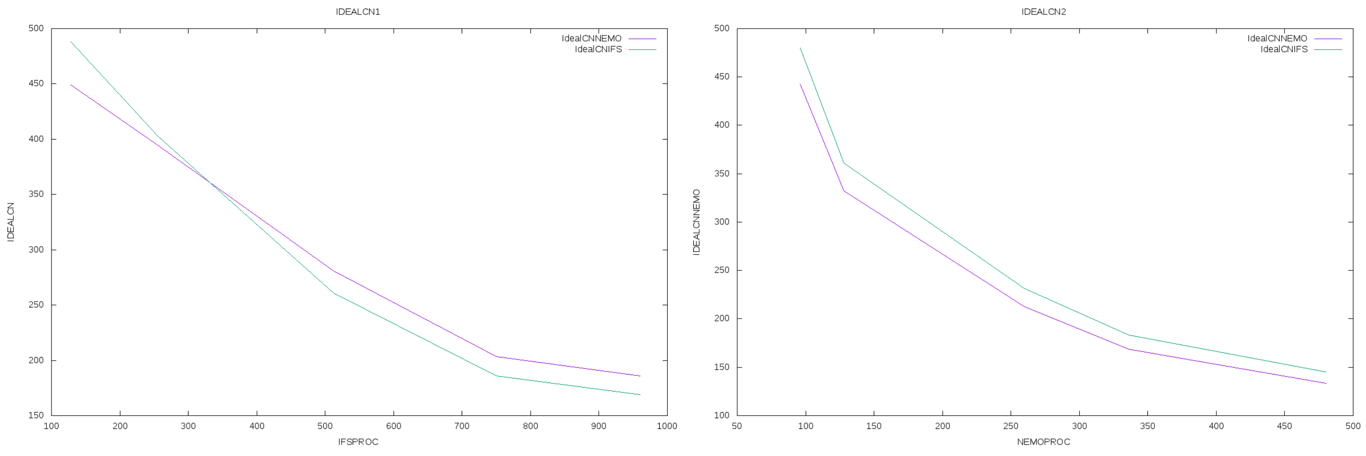


Figure 14: *IdealCN1 and IdealCN2*

Once again, the best number of processors seems to be beyond 450 for NEMO and probably 900 for IFS. Unfortunately, IFS and NEMO aren't totally balanced. The percentage of unbalance between RTS IFS and RTS NEMO is given this time by the unbalance metric:

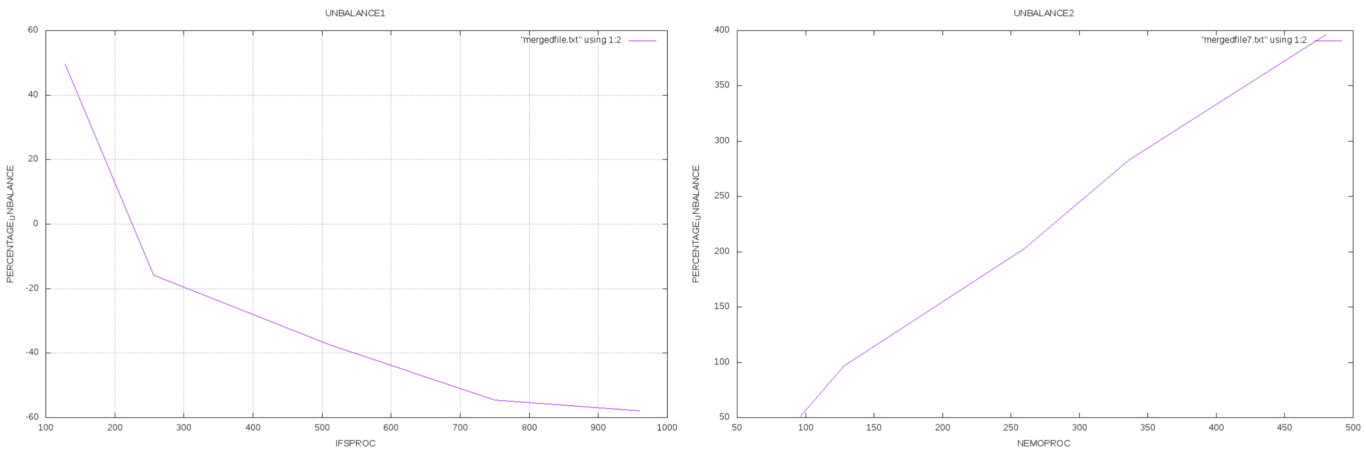


Figure 15: *unbalance1 and unbalance2*

Here we have an acceptable difference of values. In fact the values for IFS unbalance are negative and those for NEMO unbalance are positive. At the end, the sum is not too important. Let's check for Mintotal, the fewer time obtainable:

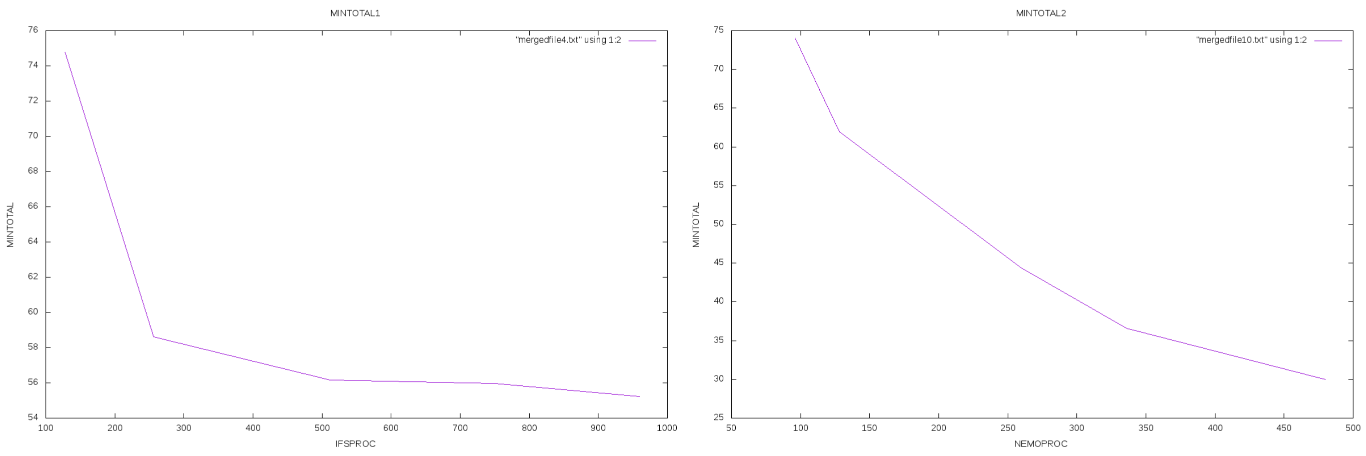


Figure 16: *MinTotal 1 and MinTotal2*

The fewer time that we can obtain is decreasing with the number of processors. Clearly, we can assume that the previous results are good. If we check the waiting time now we have for IFS and NEMO:

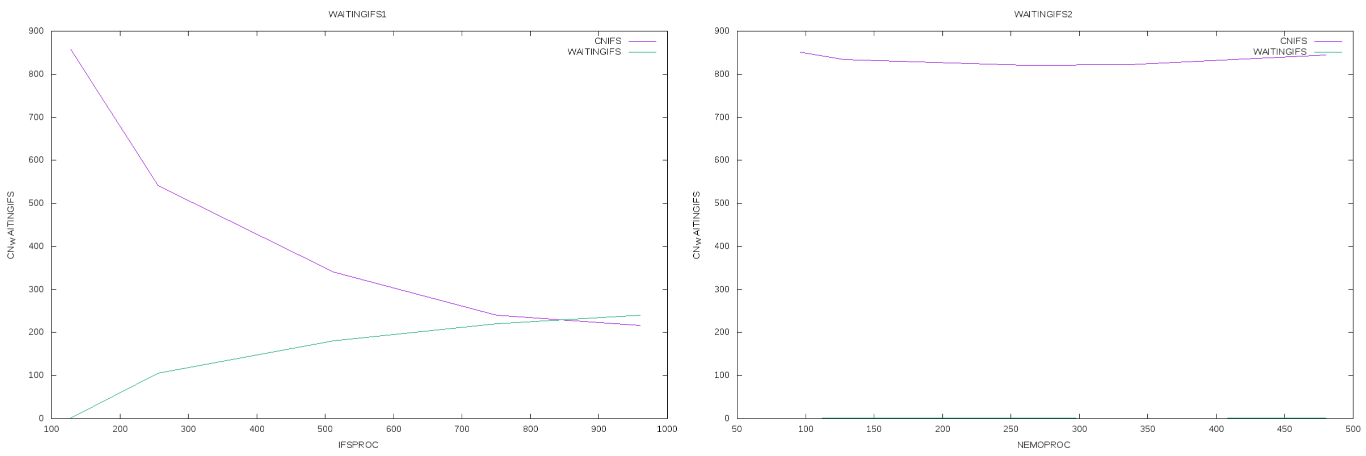


Figure 17: *WAITINGIFS1 and WAITINGIFS2*

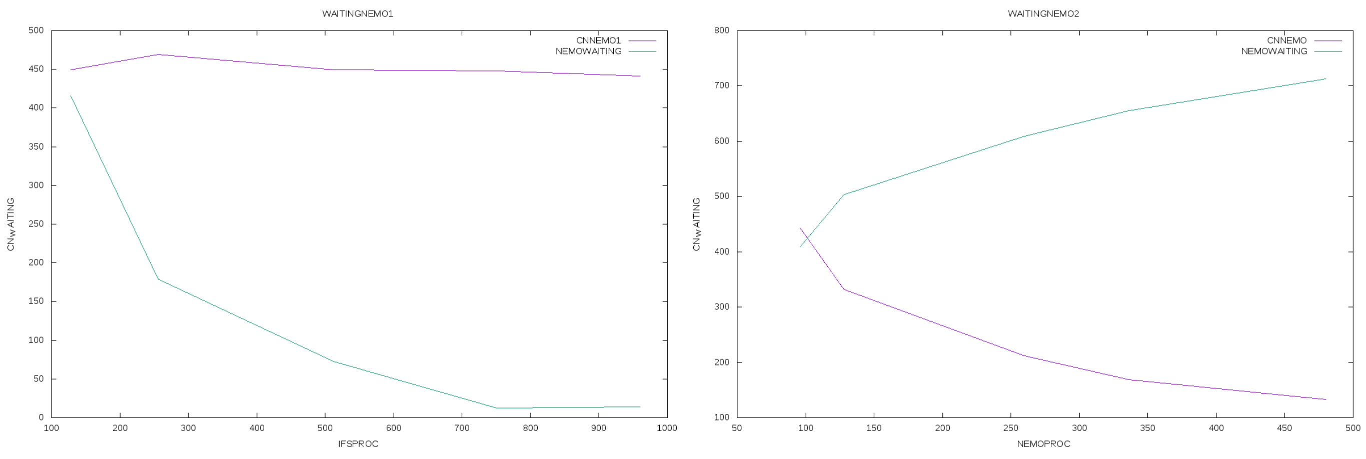


Figure 18: *WAITINGNEMO1 and WAITINGNEMO2*

Here we can see that IFS is almost not waiting for NEMO and that NEMOWAITING is decreasing with the raise of IFS' number of processors. Finally, let's check the simulated years per day:

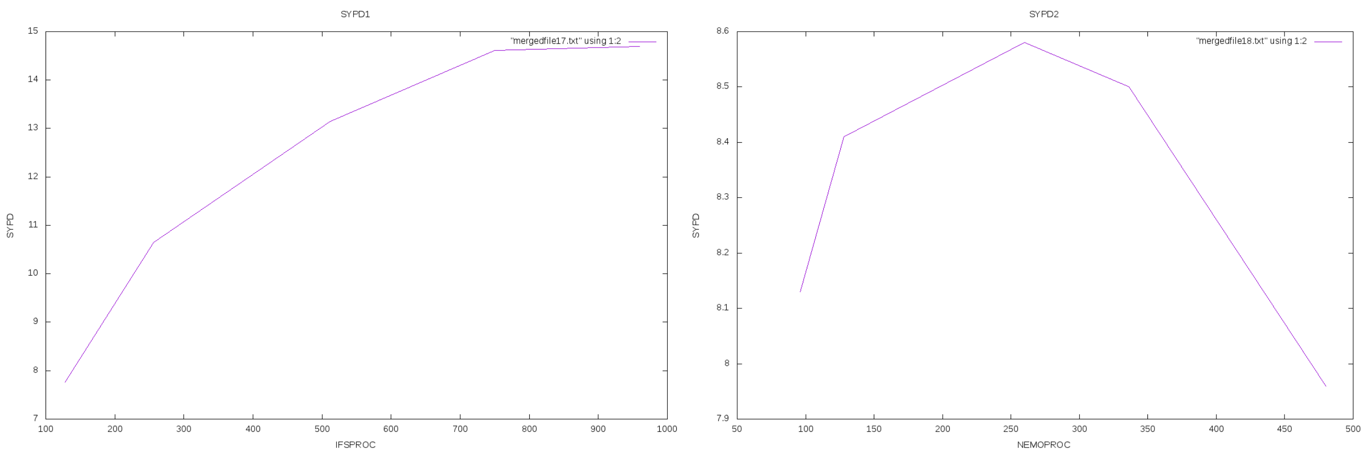


Figure 19: *SYPD1 and SYPD2*

The biggest number is obtained for the number of processors given before for IFS. For NEMO it's not the case but the difference of years simulated between a low number of processors and the one selected is very low.

Conclusion: We can assume that the best number of processors is 960 for IFS and 480 for NEMO.

6.3 The improvement of the implementation: Time and Energy to solution methods

All the previous results were obtained manually and it took some time to understand which was the best combination of processors between IFS and NEMO. In the purpose of being more efficient, an automating of this process was required. Thereby I created new metrics (cf part 4.5) and I wrote a new file: "*scalability_options.sh*". Thus, two methods were implemented: the **Time to solution** and the **Energy to solution**. The main ideas are the next ones:

- To begin, **we need to launch a classic project to have some results**.
- Once it's done, we need to take the RTS values and then select the number of processors corresponding to the **best value of RTS for IFS only.** We also save the value of the efficiency in a new variable. Thus, we need to launch the experiment again but this time with this chosen value.

- Then we launch an experiment again:
  - For the **Time to solution**, we select the previous number of processors for IFS and the number of processors of NEMO corresponding to the **best value of RTS NEMO**.
  - For the **Energy to solution method**, we consider the same value for IFS but the value of NEMO corresponding to the **best unbalance.** We also save in a new variable the **efficiency** corresponding to this value of the unbalance.
- Finally, **we launch the experiment a third time** with the best value of processors for NEMO and IFS in the same time. Thanks to the previous launch, we have new values of RTS for IFS and NEMO. We need then to take a **look at the RTS or unbalance values again** and do the same processes. However, for the time to solution, if the RTS NEMO value obtained at the step t+1 is bigger than at the step t then we need to stop and for the energy to solution, if the efficiency is lower than 75% we need then stop too.

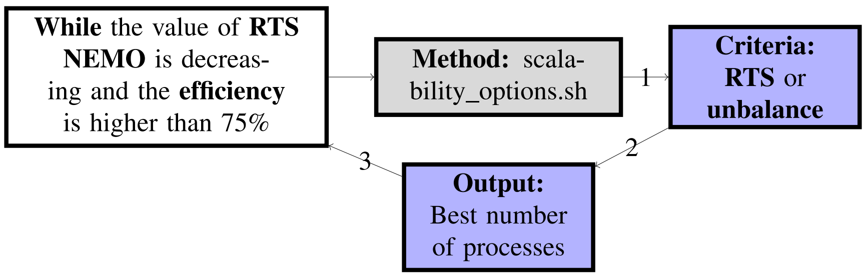

While the value of RTS NEMO is decreasing and the efficiency is higher than 75%
Method: scalability_options.sh
1
Criteria: RTS or unbalance
2
Output: Best number of processes
3

Figure 20: *The new scalability process*

Thus we obtain for the time to solution **960** processors for IFS and **480** for NEMO. For the energy to solution the results are **960** processors for IFS and **336** for NEMO.

In conclusion, in this part I wrote **7** reports, created **9** bash scripts and modified a Python file for two different purposes.

7 Load balance test

7.1 Classical implementations

Once all the results obtained in the previous part, it was decided to improve the load balance [5]. The load balance improves the distribution of workloads across multiples resources. It aims to optimize the resource use, maximize throughput and minimize the time of response. As it is explained in the section (4.4), most of the time NEMO is waiting for IFS. This time is given by the metric unbalance. The purpose of this part has been to obtain automatically the best number of processors for IFS and NEMO considering the value of RTS IFS, RTS NEMO and with the lowest possible unbalance value at the same time.

First, I had to think about a way of implementation. Thus I wrote the eighth report "*load unbalance*" relying on the idea found and created the file "*unbalance.sh*" corresponding to the **principal method**:

- To begin, **we need to launch a classic project to have some results**.
- Once it's done, we need to take the RTS values and then select the number of processors corresponding to the **best value of RTS for IFS only.** We also save the value of the efficiency in a new variable. Thus, we need to launch the experiment again but this time with this chosen value.

- With the previous project, we have one possibility to find the unbalance value. In fact, it's the only way to have access to the good unbalance value because the number of processors for IFS is constant. We then write the value of the processors chosen and of the unbalance in a new file. At the end, we can **select the number of processors for NEMO** corresponding to the minimal unbalance value and we launch the experiment again.
- Thanks to the previous launch, we have new RTS values for IFS and NEMO. We need then to take a **look at the RTS values again** and do the same process. However, if the RTS NEMO value obtained at the step t+1 is greater than at the step t then we need to stop. If the value of the unbalance is less than 5 percent, we need to stop.

Secondly, I had to go deeper and propose a new method that I called **the deep load unbalance**. Accordingly, I wrote the report number nine "*Deep load unbalance: file Unbalance_knownvalues.sh*" and created one bash scripts "*Unbalance_knownvalues.sh*". This method which corresponds to the **first method** adds accuracy. In fact, the previous method is giving the best number of processors in a first part (which is the scalability test actually) and then the combination between the best number of processors and the unbalance value in a second part (load balance test). But, let's suppose that we already have done the scalability test, we can proceed differently. In fact, we know the best combination of processors between IFS and NEMO so we just have to take a look at the unbalance values for some IFS and NEMO neighboring values. This adds accuracy because the table used for the scalability test is composed with the extended values (table_NEMO=("96" "128" "260" "336" "480")) when the one used here contains close values (example for NEMO=**480**, step=20, table_NEMO=("460" "470" "**480**" "490" "500")). However, an idea could have been to use a huge table directly in the file "*unbalance.sh*" but the time of computation would have been too long. Thus, with the previous method we can have results for closest values and we are quite sure that the result obtained is good:

- First, we need to **ask the user for the values**.
- After that we can **launch the experiment**. Thus, we launch the experiment with a new table containing always the same value for IFS (example: 960) and very close values for NEMO (example: TABLE= ("480-value" "480" "480+value")) to be sure that the unbalance value find for the best combination of processors is really the good one.
- Once it's done, as usual, we need to **monitor**, to **launch the results** (file ec-earth_autosubmit_results.sh) and to **display** (file ec-earth_autosubmit_perf_display.sh).
- Finally, we must **take the best unbalance value** and print this value as well as the number of processors for IFS and NEMO in one file ("*results_knownvalues.txt*").

$\rightarrow$ **Remark**: To be even more accurate, we can use the file "*average_three.sh*" introduced in the scalability section that launches a project three times and gives the average.

7.2 The improvement of the implementations: Time and Energy to solution methods

To have more information the unbalance metric is not the only one considered anymore. In fact, two methods where added to the files "*unbalance.sh*" and "*Unbalance_knownvalues.sh*" and proposed to the user: the time to solution and the energy to solution. The tenth report "*Time and energy to solution methods*" was also written. The principle of those methods are not exactly the same that on the scalability part.

- In the **Time to solution** method, the requirement is the **minimum total execution** time. This correspond to the metric **unbalance8** (in the scalability part we were considering the value of RTS NEMO). Thus, we select the number of processors for NEMO corresponding to the lowest value for this unbalance8.

- In the **Energy to solution** method, the requirement is the **minimum time possible for the load unbalance**. The metrics taken into consideration are the **unbalance** and the **efficiency**. We select the number of processors for NEMO corresponding to the best value of the unbalance and we make sure that the corresponding value of the efficiency is good (greater than 75%).

7.3 Derived implementations

A **second method** has been proposed following the file "*Unbalance_knownvalues.sh*". Thus, the file "*Unbalance_knowntables.sh*" was created and the reports number eleven, twelve and fourteen "*Comparison of the unbalance results*", "*Deep load unbalance: file Unbalance _knowntables.sh*", "*Other unbalance cases*" were written. In fact, in the previous Bash script, the number of processes is increased for NEMO until we find a good value for the **unbalance**. However, it's not always the best way to improve the combination between the models. This is why the second file aims to compare different things regarding the method. For the **time to solution method** different possibilities are existing. In fact, like it was explained in the section (4.4) of the report, we can have eight steps for each model and the length of the steps are different. Thus, regarding the fact that the regular steps, the fourth step or the eighth step is more important for IFS or NEMO, the value took into consideration should be the **unbalance**, **unbalance4** or **unbalance8** and not always the unbalance8. Regarding the results we should also increase or decrease one model. Thereby we have 9 possibilities.

$\rightarrow$ **Example**: The computation time of the fourth step of NEMO is higher than for IFS, we should then $\uparrow$ the number of processors for NEMO or $\downarrow$ the number of processors for IFS.

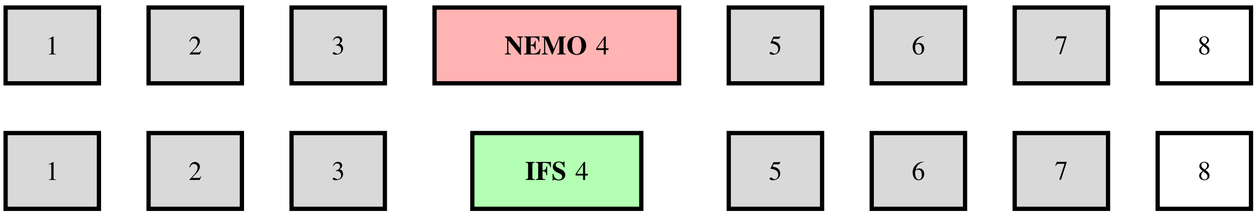


Figure 21: *Example -> NEMO4 > IFS4*

For the **energy to solution**, we consider the values of **CN** and **IDEALCN** for both models. Then the file selects the model which has the bigger difference between those two variables. At the end, if the difference is positive then we increase the number of processes for the previous model and if it's negative, we decrease it.

In addition, the **third method** created, the file "*Unbalance_method3.sh*", is just a combination of the two files "*unbalance.sh*" and "*Unbalance_knownvalues.sh*". We use the number of processes returned by the first file as an input for the second one. We could also use the file "*Unbalance_knowntables.sh*" as second file.

Finally, a **fourth method**, "*unbalance_final.sh*" has been developed. First, the user needs to enter the values of the number of processors for IFS and NEMO. Then, a project is launched and two choices are proposed. For the **time to solution**, we take the number of processors for NEMO corresponding to the lowest value of **RTS_NEMO**. Then we launch a project again with this number of processors for NEMO and we select the number of processors for IFS which corresponds to the lowest value of the **unbalance8** lower than 5% if this value exists or to the lowest value of **RTS_IFS**. **If** RTS_NEMO < RTS_IFS we decrease NEMO processes and we take the number of processors for NEMO corresponding to a value of the **unbalance** < 5 % if this value exists or corresponding to the lowest value of RTS_NEMO so that RTS_NEMO > RTS_IFS. Thus, we launch a project a third time with this number of processors for NEMO and the previous one for IFS to have the complete results. For the **energy to solution**, we take the number of processors for NEMO corresponding to the **efficiency** > 75 % so that at the efficiency of the next number of processors in the table < 75 %. Then we launch a project again with this number of processors for NEMO and we select the number of processors for IFS corresponding to the value of the **unbalance8** < 5% if this value exists or the **efficiency** > 75% so that the efficiency of the next number of processors in the table < 75 %. Finally, we use the same 9 possibilities that in the file "*Unbalance_knowntables.sh*" to decide to increase or decrease IFS or NEMO and we launch the project a third time. The final number of processors selected for NEMO (and IFS if the direction was the increase of this model) corresponds to the best value of the **unbalance** < 5% if it exists or to the **efficiency** > 75% so that the efficiency for the next number of processors in the table < 75%.

Thus, to present all the results introduced in the previous sections, a fifteenth report was written, "*Complete results for the scalability and load balance parts*".

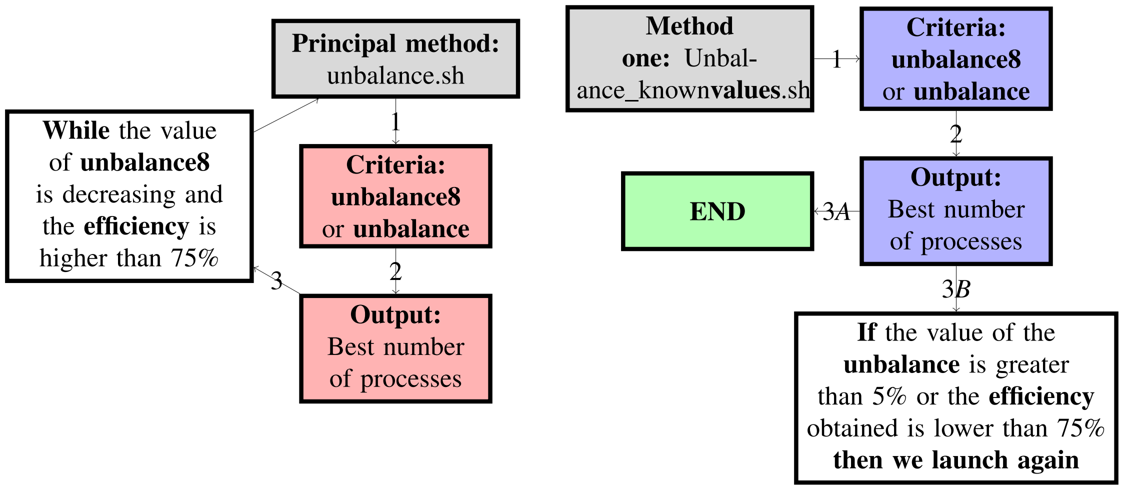

Principal method: unbalance.sh
1
Criteria: unbalance8 or unbalance
2
Output: Best number of processes
3
While the value of unbalance8 is decreasing and the efficiency is higher than 75%
Method one: Unbalance_knownvalues.sh
1
Criteria: unbalance8 or unbalance
2
Output: Best number of processes
3A
END
3B
If the value of the unbalance is greater than 5% or the efficiency obtained is lower than 75% then we launch again

Figure 22: *Two first possibilities of use*

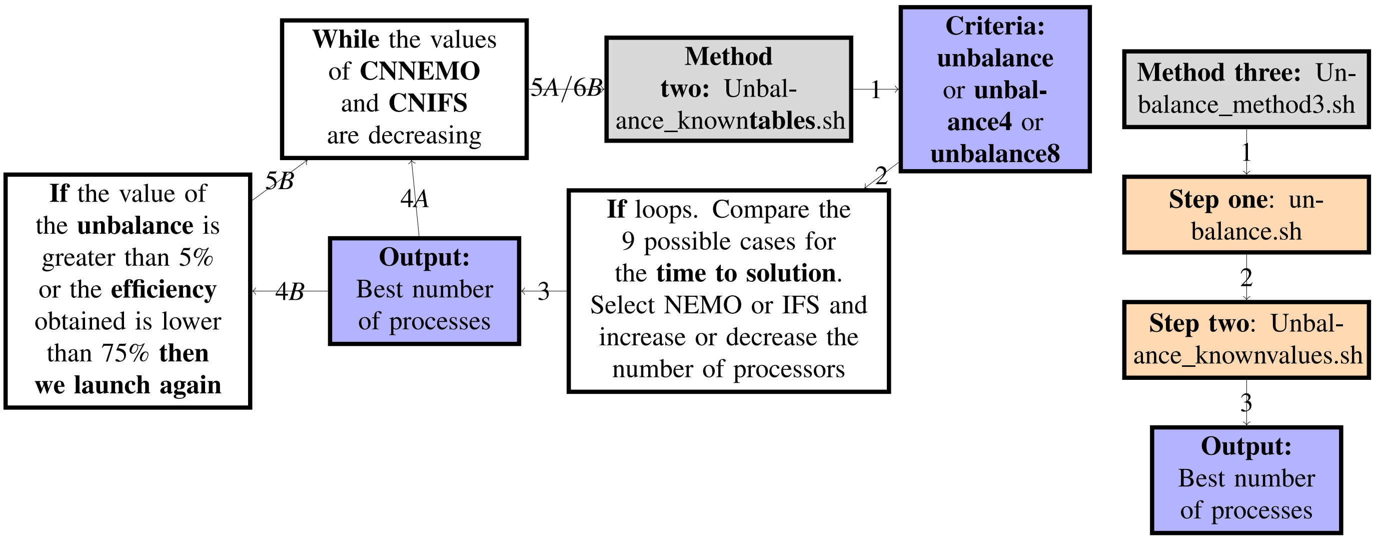

While the values of CNNEMO and CNIFS are decreasing
5A/6B
Method two: Unbalance_knowntables.sh
1
Criteria: unbalance or unbalance4 or unbalance8
2
If loops. Compare the 9 possible cases for the time to solution. Select NEMO or IFS and increase or decrease the number of processors
3
Output: Best number of processes
4A
4B
If the value of the unbalance is greater than 5% or the efficiency obtained is lower than 75% then we launch again
5B
Method three: Unbalance_method3.sh
1
Step one: unbalance.sh
2
Step two: Unbalance_knownvalues.sh
3
Output: Best number of processes

Figure 23: *Two other possibilities of use*

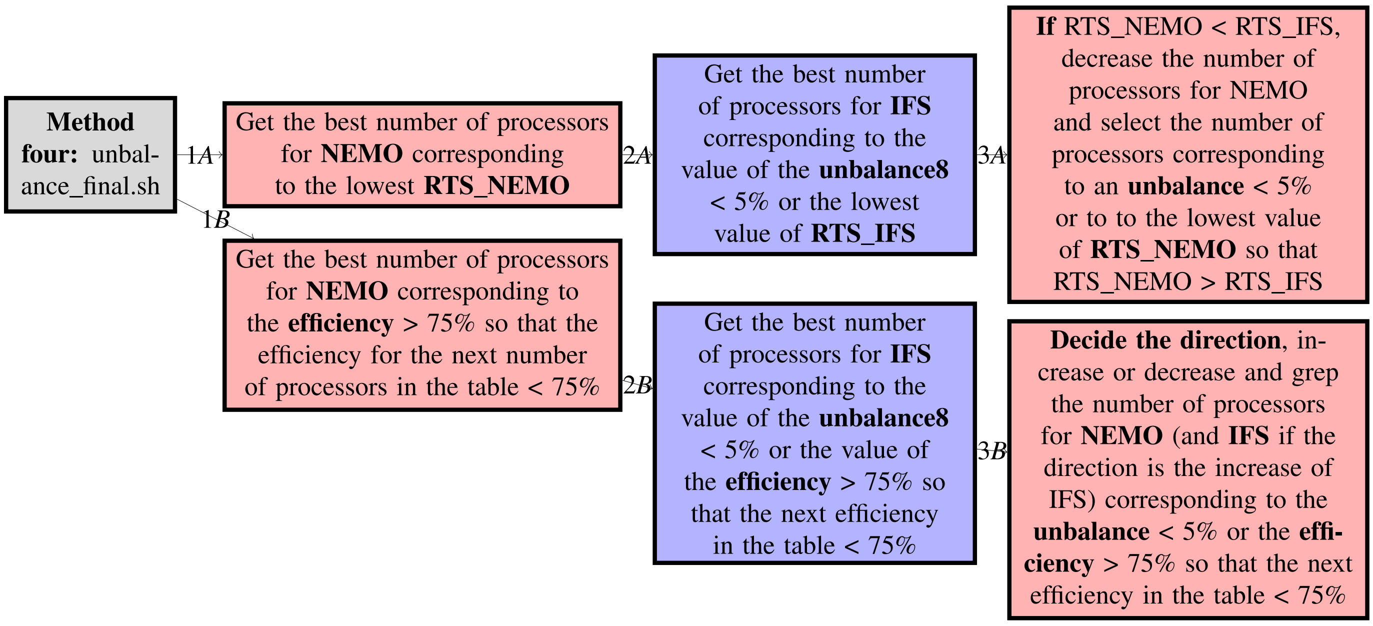

Method four: unbalance_final.sh
1A
1B
Get the best number of processors for NEMO corresponding to the lowest RTS_NEMO
Get the best number of processors for NEMO corresponding to the efficiency > 75% so that the efficiency for the next number of processors in the table < 75%
2A
2B
Get the best number of processors for IFS corresponding to the value of the unbalance8 < 5% or the lowest value of RTS_IFS
Get the best number of processors for IFS corresponding to the value of the unbalance8 < 5% or the value of the efficiency > 75% so that the next efficiency in the table < 75%
3A
3B
If RTS_NEMO < RTS_IFS, decrease the number of processors for NEMO and select the number of processors corresponding to an unbalance < 5% or to to the lowest value of RTS_NEMO so that RTS_NEMO > RTS_IFS
Decide the direction, increase or decrease and grep the number of processors for NEMO (and IFS if the direction is the increase of IFS) corresponding to the unbalance < 5% or the efficiency > 75% so that the next efficiency in the table < 75%

Figure 24: *Last possibility of use*

In conclusion, in this part I adapted an old Python file, I wrote **6** reports and created **5** bash scripts.

7.4 Comparison of the results

Let's consider these tables for IFS and for NEMO:

table_**IFS**=("128" "256" "512" "750" "960") and table_**NEMO**=("96" "128" "260" "336" "480")

We know, thanks to the **scalability part**, that the best number of processors is equal to **960** for IFS and **480** for NEMO.

The **principal method** gives us two different results depending of the chosen method. For the **time to solution** method, we find that the best number of processors is **960** for IFS and **96** for NEMO. For the **energy to solution** method, the better solution seems to be **960** for IFS and **336** for NEMO. Probably, it's not the best combination and we will have a better value by applying the second method.

The **first method** is made to give the best combination of processors regarding the method used (Time or Energy) with an extended table. However, the principle is to use this file when you want to launch a project with a table compounded of values of NEMO centered around a chosen value. In this case, we can apply the previous values obtained with the file "*unbalance.sh*". Thus, this is equal to the use of the **third method**. If we use the first method, we obtain for the **time to solution** method **960** for IFS and **480** for NEMO. For the **energy to solution** we find a number of processors equal to **960** for IFS and **500** for NEMO.

The **second method** indicates that for an input of 960 processors for IFS and 480 for NEMO, the best combination for the **time to solution** and for the **energy to solution** is the increase of the number of processors for IFS giving **975** for this previous one and **480** for NEMO for both methods.

The **fourth method** showed that for the **time to solution**, the best number is equals to **960** for IFS and **260** for NEMO. For the **energy to solution**, it seems like the values are **256** for IFS and **336** for NEMO.

Figure 25: *Presentation of the results*

Méthode	IFS	NEMO	CUnbal	Unbal8	Efficiency	RTS_IFS	RTS_NEMO
Principal with TTS	960	96	x	13.32	1.0	x	x
Principal with ETS	960	336	1.557	x	x	x	x
One/three with TTS	960	480	x	93.03	0.985	x	x
One/three with ETS	960	500	3.735	x	x	x	x
Two with TTS	975	480	x	159.4	x	x	x
Two with ETS	975	480	18.87	x	1.041	x	x
Four with TTS	960	260	200.0	522.9	x	0.196	0.216
Four with ETS	256	336	47.42	218.6	1.067	x	x

Thus, for the scalability best combination of processors (960 for IFS and 480 for NEMO), in a first part, it's seems like the speed up is increasing with the number of processors for IFS and for NEMO. This is normal since we are increasing the number of processors. We can assume that the speed up is super linear since the models are really low at the beginning.

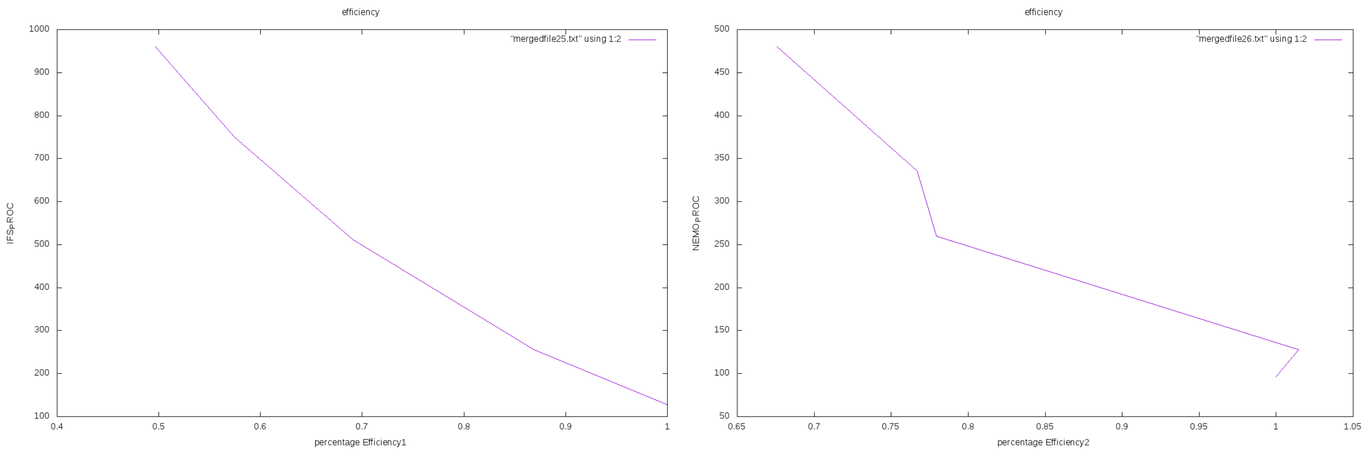


Figure 26: *The efficiency*

If we take a look to the efficiency, we can see that it's decreasing for IFS and for NEMO. Those results are logical since the efficiency is proportional to the opposite of the speed up. However, it's also logical because the efficiency of the lowest value of the number of processors is equal to 1.

8 Graphical interface

The final part of the project was dedicated to the presentation of the results. The programming language used is **Python** and more precisely the third version of it. To present the results, I wrote consequently the thirteenth report, "*Graphical interface*" and I created a new Python script "*graphical_interface.py*".

First, the user has to **choose between the scalability and unbalance part**. Then, for both parts, he has the possibility to choose to **read the documentation**, to **launch one of the possible methods** or to **exit**. The documentation part includes all the reports written in this internship. Thus, if the user wants to read one of them, the interface will open the pdf file.

Once the user finished to read the documentation, he can **launch one of the methods of the previous chosen part** or **Exit**.

Conclusion

One of the main goals of my work was to achieve a **good scalability** of the EC-Earth model running on different configurations, so that I could evaluate the best efficiency of the execution of EC-Earth for two approaches, time to solution (the best one in terms of execution time) and energy to solution (the best one in terms of unbalance). An other was to create new metrics to explore the **load balance** issue with those two approaches adapted to this part. Finally, I had to create a **graphical interface** to present the results in a better way.

To do so, I had first to read some technical reports and understand how a project is working. Thereby, I wrote the first report *"Agnes files documentation"*.

Then I worked on the **scalability part**. The purpose of this part was to find the best combination of processors between IFS and NEMO to have the minimal computation time possible. I started from scratch, trying to make links between the files and gathered all the required folders. Thus, I began to debug. The main problems in this part were to understand the among of files/folders furnished, the "autosubmit" errors and to find which file should be debugged. When I succeeded to launch some projects, I wrote consequently two reports, "*Study case: 128 processors for NEMO and IFS*" and "*Study case: 480 processors for NEMO and 960 processors for IFS*". Thereafter, I wrote "*The complete documentation for the creation of a project*" to explain how a project must be created starting from the very beginning since no complete documentation was available. I also created eight other Bash scripts to automatize the processes. The plots with the metrics obtained, it was decided then to create a new metric. In this purpose I modified a Python file and wrote the seventh report, "*New metric: Speed up*". Then added two methods, Time and Energy to solution methods, leading to addition of a new metric in the Python file, the efficiency and to the creation of an other Bash script "*scalability_options.sh*".

Afterwards, I started the **load balance part**. The purpose of this part was to go even further. In fact, finding the best number of processors gives a good value for the computational time (CN and RTS) of each model alone but do not assure a good unbalance. This unbalance metric corresponds to the waiting time between IFS and NEMO. Thus reducing the unbalance equals to having almost the same time steps for each models and thereby reducing the computational time of the coupled model. Thereby, I presented my results through the eighth report, "*load unbalance*" and created the **principal method**, "*unbalance.sh*". I also modified the previous Python file to add new metrics: RTS at step four and eight. Two others methods, such as in the previous part, were integrated just after leading to the writing of an other report called "*Time and Energy to solution methods*". To go further I created two other files, the **first method**, "*Unbalance_knownvalues.sh*", the **second method**, "*Unbalance_knowntables.sh*" and I wrote two reports. In addition, a **third method** was made to launch the principal and then the first method: "Unbalance_method3.sh". Finally a **fourth method**: "*unbalance_final.sh*" was created giving the best combination of the previous files. However, the other files remains necessary. Thereby, a fifteenth report was introduced to compare all the results. Here it was also difficult to think about an implementation that deals with all the possible exceptions. I had to start and then improve the files until it was satisfying enough.

At last, a graphical interface was created in Python 3 using **easygui**, **os** and **sys** libraries. Consequently I created the file "*graphical_interface.py*" and I wrote the thirteenth and last report "*Graphical interface*".

To conclude, in this 10 weeks internship, I created **15** scripts in Bash and Python and I wrote **15** reports. Regarding the scripts, the main inconvenient is that almost all files were created to launch a table of 5 values for each model (table_**IFS**=("128" "256" "512" "750" "960")). This table is more that enough and is the convention in Bsc. However, this is something that could be improved in the future.

Bibliography

[1] Barcelona Supercomputing Center, "Barcelona Supercomputing Website." (https://www.bsc.es).

[2] Barcelona Supercomputing Center, "Logo Supercomputing Center." (https://www.trust-itservices.com/portfolio/partnership/bsc-barcelona-supercomputing-center).

[3] Blaise Barney, Lawrence Livermore, "Parallel Computation." (https://computing.llnl.gov/tutorials/parallel_comp/#Whatis).

[4] Eric Maisonnave, Arnaud Caubel, "Lucia, load balancing tool for oasis coupled systems," 2017.

[5] Mario Acosta, "Performance analysis of EC-Earth 3.2: Load balance," 2017.

[6] Mario Acosta, Xavier Yepes-Arbos, Sophie Valcke, Eric Maisonnave, Kim Serradell, Oriol Mula-Valls, Francisco Doblas-Reyes, "Performance analysis of EC-Earth 3.2: coupling," 2016.

[7] Polytech Sorbonne, "Logo Polytech Sorbonne." (https://www.facebook.com/pg/PolytechParisUPMC/photos/?tab=album&album_id=156421134394814).

[8] Sorbonne Université, "Logo Sorbonne Université." (http://logonews.fr/2018/01/04/nouvelle-fusion-sorbonne-universite-presente-logo/).

Annex

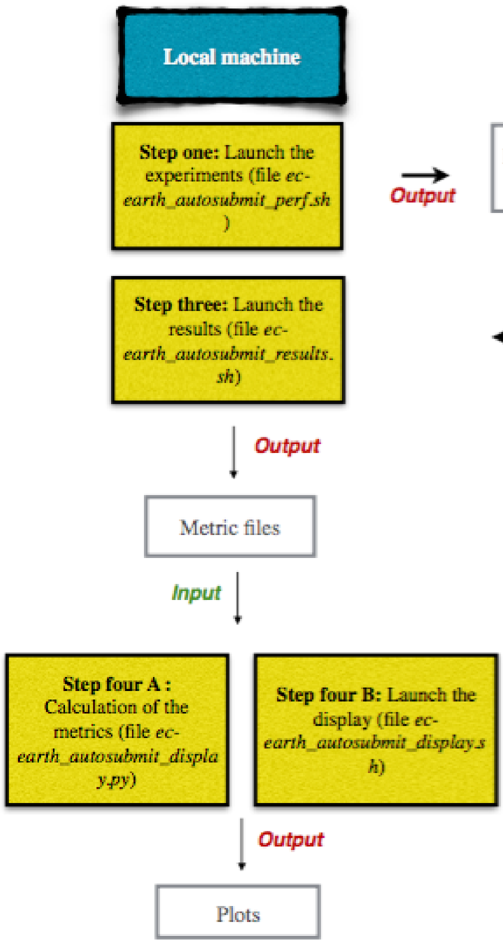

Local machine
Marenostrum
Step one: Launch the experiments (file ec-earth_autosubmit_perf.sh)
Output
Experiments folders
Input
Step two: Experiments running
Output
Lucia files
Input
Step three: Launch the results (file ec-earth_autosubmit_results.sh)
Output
Metric files
Input
Step four A : Calculation of the metrics (file ec-earth_autosubmit_display.py)
Step four B: Launch the display (file ec-earth_autosubmit_display.sh)
Output
Plots

Figure 27: *Organization of the first section of the scalability part*

The first step is the launch of the file ec-earth_autosubmit_perf.sh. This file has the purpose to create all the **experiments folders** presented at the part 5.1. This folders are created locally and on Marenostrum at the same time. Once the experiments created, they are also **running**. This means that Marenostrum is gathering its resources to do the work. The result is multiple. In fact, when this work is finished, some files have been created locally such as the reports of the advance of the work. However, it's not all, some other files are created on Marenostrum such as **Lucia files**. As seen in the 4.3 part, those files are really important since they are giving the computational time of each step for each model. The next task is then to copy them locally. Once it's done, they are taking as input by the file ec-earth_autosubmit_results.sh. This file is made to treat the Lucia files and give the principal **metric files**. Those last ones are the files representing the results of some metrics explained in the part 4.5. Furthermore, they are taken at their turn as input by the file ec-earth_autosubmit_display.sh. First, the Python file ec-earth_autosubmit_display.py is calculating the metrics that are obtained from the principal metrics. Then the Bash file ec-earth_autosubmit_display.sh is displaying the results and creating all the **plots**.

Summary of the purposes of the main files

For the **Scalability part**, I have two methods. The first one is the classical one, here is the second:

- File **scalability_options.sh**: Going to find the best combination automatically.
  - In the **time to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. The value of the efficiency is also saved in a new variable. For NEMO, the best number of processors is also given using the best value of RTS_NEMO.
  - In the **energy to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. The value of the efficiency is saved in a new variable. For NEMO, the best number of processors is given using the best unbalance.

    I'm doing that while the value of CN_NEMO is decreasing and the value of the efficiency is greater than 75%.

For the **Unbalance part**, I have five methods:

- File **unbalance.sh**: Going to find the best combination automatically.
  - In the **time to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. The value of the efficiency is also saved in a new variable. For NEMO, the best number of processors is giving using the best unbalance8.
  - In the **energy to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. The value of the efficiency is also saved in a new variable. For NEMO, the best number of processors is giving using the best unbalance. The corresponding value of the efficiency is also given and saved in a variable.

    I'm doing this while the efficiency is greater than 75% and the unbalance8 is decreasing.

- File **Unbalance_knownvalues.sh**: Takes the best combination as input. Uses a specific table.
  - In the **time to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. For NEMO, the best number of processors is giving using the best unbalance8.
  - In the **energy to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. For NEMO, the best number of processors is giving using the best unbalance. The corresponding value of the efficiency is also given and saved in a variable.

    There is no while loop. However, there is a if loop. If the value of the unbalance is greater than 5% or the efficiency lower than 75%, we launch the experiment one more time.

- File **Unbalance_knowntables.sh**: Takes the best combination as input. Find the model to be increased or decreased and the main criteria that should be used (unbalance, unbalance4 or unbalance8). Uses a specific table.
  - In the **time to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. For NEMO, the best number of processors is giving using the best unbalance, unbalance4 or unbalance8.

  - In the **energy to solution**, I'm taking the best value of processors for IFS comparing the RTS_IFS values. For NEMO, the best number of processors is giving using the best unbalance. The corresponding value of the efficiency is also given and saved in a variable.

    I'm doing this while the values of CNNEMO and CNIFS obtained at the step t+1 are lower than those at the step t.

- File **unbalance_method3.sh**: Launches the file "*unbalance.sh*" and then the file "*Unbalance_knownvalues.sh*".
- File **unbalance_final.sh**: Takes a number of processors for IFS and NEMO as input. Find the model to be increased or decreased and the main criteria that should be used (unbalance, unbalance4 or unbalance8). Uses a specific table.
  - In the **time to solution**, I'm taking the best value of processors for NEMO corresponding to the lowest value of RTS_NEMO. Then I take the number of processors for IFS corresponding to the lowest value of the unbalance8 lower than 5% if this value exists or to the lowest value of RTS_IFS. If RTS_NEMO < RTS_IFS, then I decrease the number of processors or NEMO and I launch a project a third time. Then I take the number of processors for NEMO corresponding to the unbalance < 5% or to the lowest value of RTS_NEMO so that RTS_NEMO > RTS_IFS.
  - In the **energy to solution**, I'm taking the best value of processors for NEMO corresponding to the efficiency > 75% so that the efficiency of the next number of processors in the table < 75%. Then I take the best number of processors for IFS which corresponds to the value of the unbalance8 < 5% or the efficiency < 75% so that the efficiency of the next number of processors in the table < 75%. Finally, the file decides of the increase or decrease of IFS or NEMO and the project is launched a third time. The best number of NEMO corresponds to the best value of the unbalance < 5% or efficiency > 75% so that the efficiency of the next number of processors in the table < 75%.

$\rightarrow$ **Remark**: It's possible to use the file "*average_three.sh*" introduced in the scalability part to launch one of the files above three times and then take the average of the values to get better results. In fact, at each launch one file (example at the first launch: "*file1.txt*") is created containing all the values necessary to create the metric plots. After the three launches, the average file is making the average of the values contained into the three files and is creating the same files that the ones obtained with the file ec-earth_autosubmit_results.sh.

Graphical interface

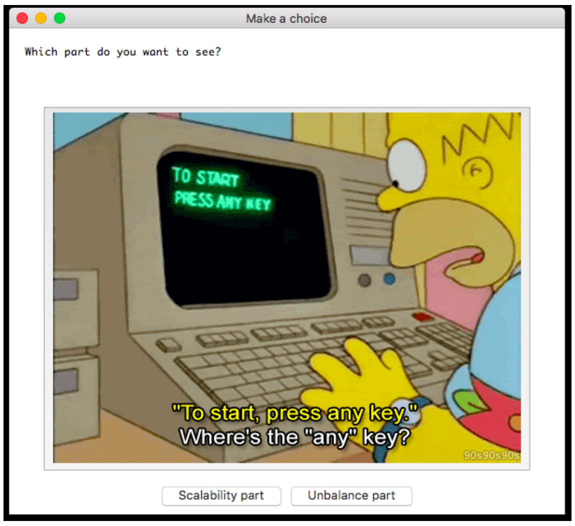


Figure 28: *Home*

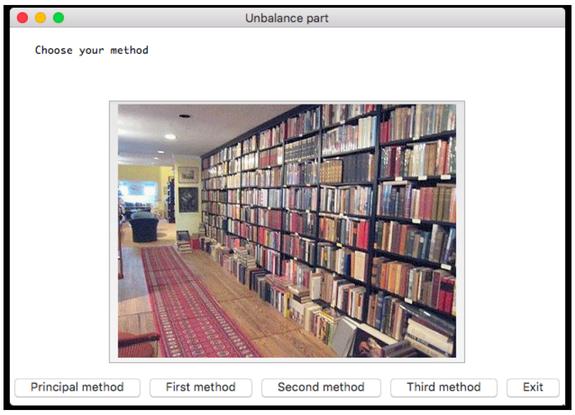


Figure 29: *Documentation*

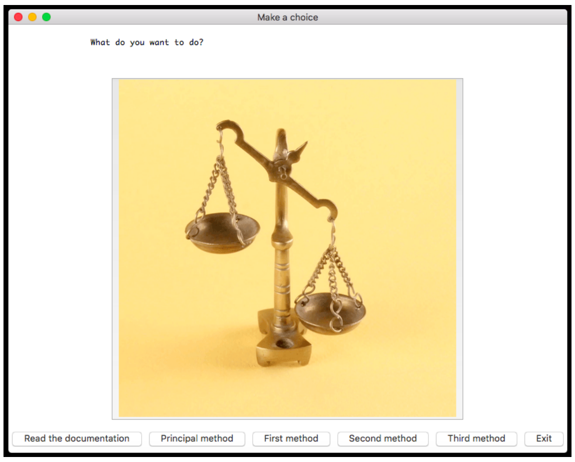


Figure 30: *Load balance part*

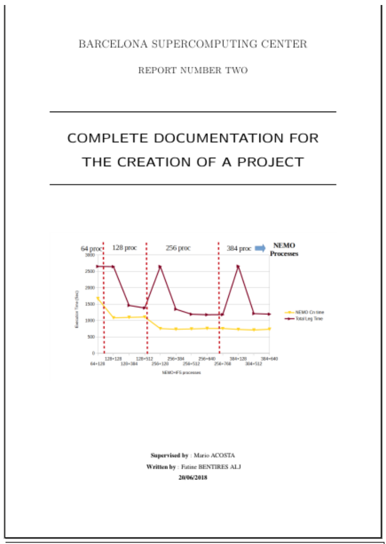
Written reports

BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER ONE

AGNES FILES DOCUMENTATION

Supervised by : Kim SERRADELL and Mario ACOSTA
Written by : Fatine BENTIRES ALJ
08/06/2018


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER TWO

COMPLETE DOCUMENTATION FOR THE CREATION OF A PROJECT

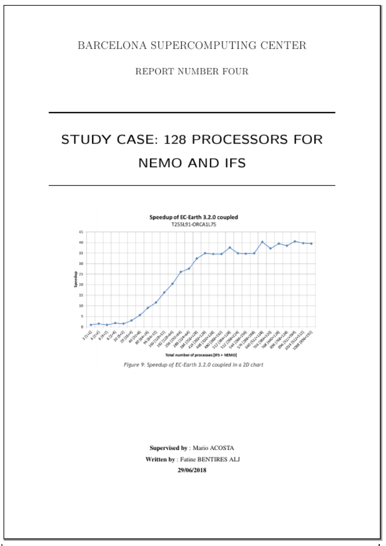
Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
20/06/2018

BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER THREE

STUDY CASE: 480 PROCESSORS FOR NEMO AND 960 PROCESSORS FOR IFS

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
20/06/2018

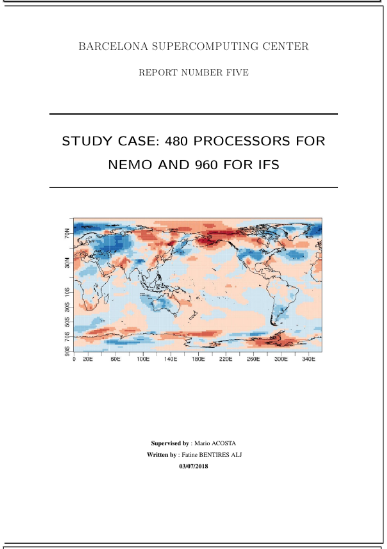

BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER FOUR

STUDY CASE: 128 PROCESSORS FOR NEMO AND IFS

Figure 9: Speedup of EC-Earth 3.2.0 coupled in a 2D chart

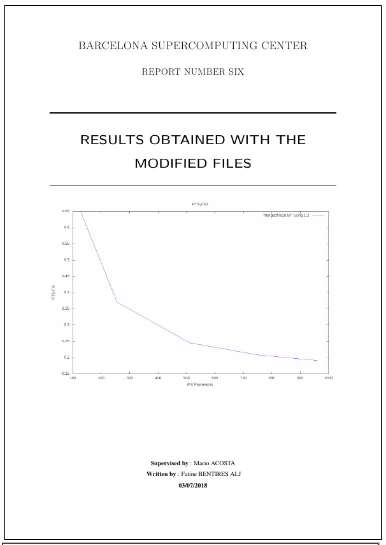
Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
29/06/2018


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER FIVE

STUDY CASE: 480 PROCESSORS FOR NEMO AND 960 FOR IFS

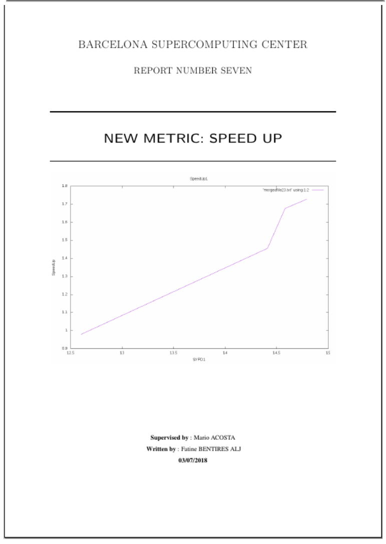
Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
03/07/2018


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER SIX

RESULTS OBTAINED WITH THE MODIFIED FILES

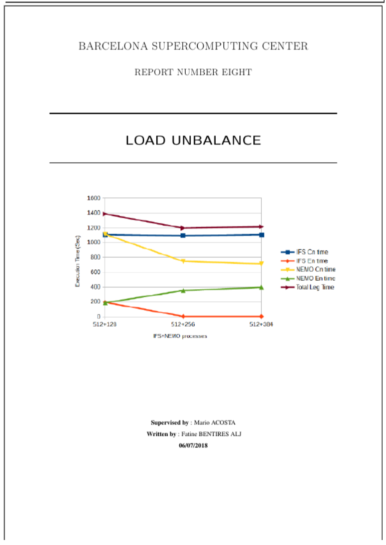
Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
03/07/2018


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER SEVEN

NEW METRIC: SPEED UP

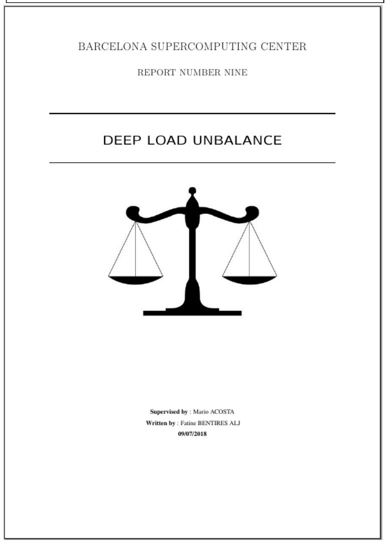
Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
03/07/2018


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER EIGHT

LOAD UNBALANCE

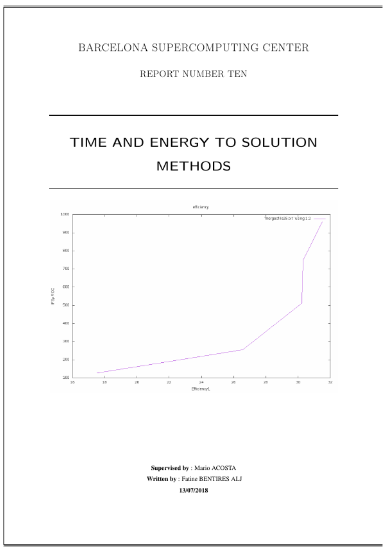
Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
06/07/2018


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER NINE

DEEP LOAD UNBALANCE

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
09/07/2018

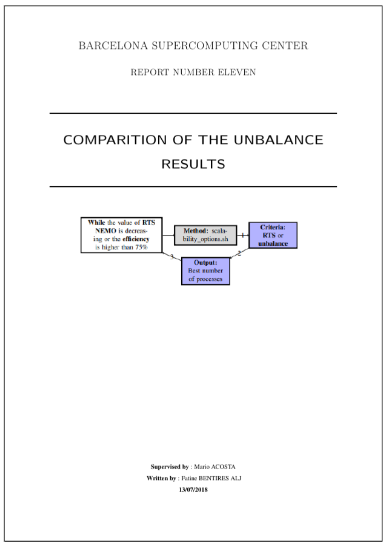


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER TEN

TIME AND ENERGY TO SOLUTION METHODS

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
13/07/2018

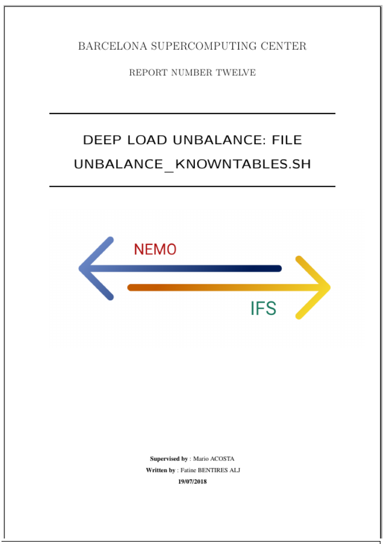


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER ELEVEN

COMPARITION OF THE UNBALANCE RESULTS

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
13/07/2018

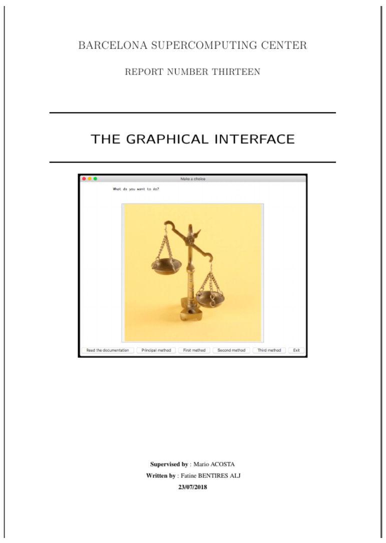


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER TWELVE

DEEP LOAD UNBALANCE: FILE UNBALANCE_KNOWNTABLES.SH

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
19/07/2018

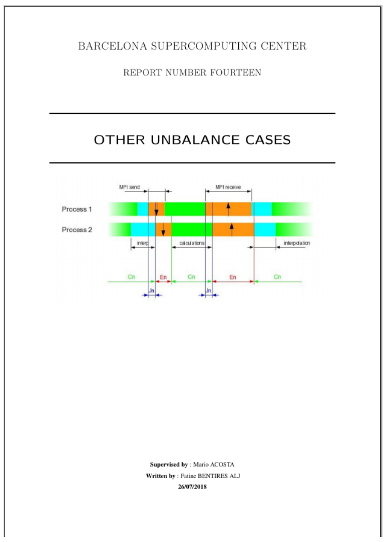


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER THIRTEEN

THE GRAPHICAL INTERFACE

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
23/07/2018

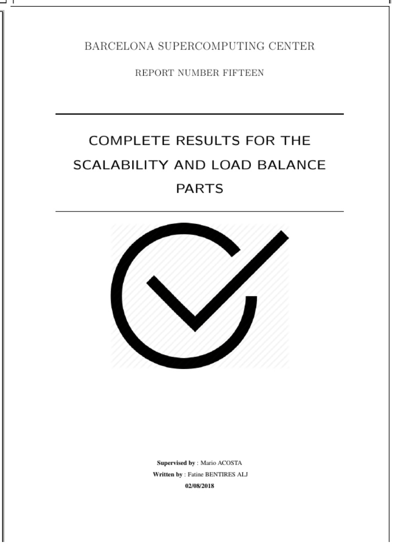


BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER FOURTEEN

OTHER UNBALANCE CASES

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
26/07/2018



BARCELONA SUPERCOMPUTING CENTER

REPORT NUMBER FIFTEEN

COMPLETE RESULTS FOR THE SCALABILITY AND LOAD BALANCE PARTS

Supervised by : Mario ACOSTA
Written by : Fatine BENTIRES ALJ
02/08/2018